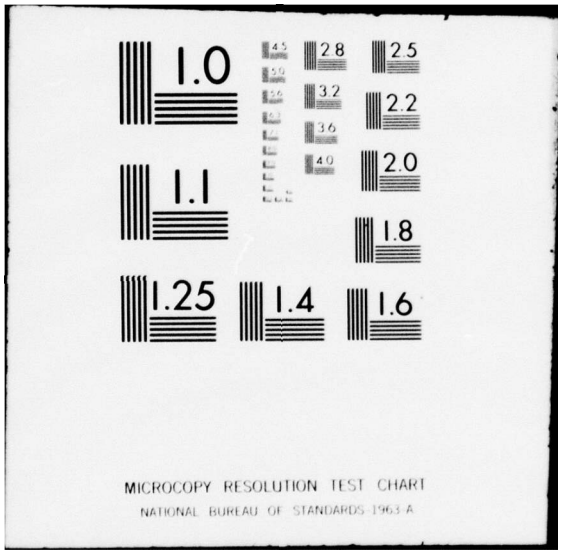
AD-A059 929 LOCKHEED-CALIFORNIA CO BURBANK F/G 9/2
SUMMARY OF 1977 INDEPENDENT RESEARCH ON USER ORIENTED REMOTE CO--ETC(U)
JUL 78 R W LINGARD, D SAIKI
UNCLASSIFIED LR-28460 NL

| OF |
AD
A059929

END
DATE
FILMED
12-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

LEVEL II

TITLE

SUMMARY OF 1977 INDEPENDENT RESEARCH ON USER ORIENTED REMOTE COMPUTING

REPORT NO.	DATE	PAGE NO.
LR 28160	7-14-78	IR

SUBMITTED UNDER (CONTRACT, SPEC., ETC.)

Form 875-3

LOCKHEED CALIFORNIA COMPANY
A Division of Lockheed Aircraft Corporation

BURBANK, CALIFORNIA, U.S.A.

LOCKHEED · CALIFORNIA COMPANY
A DIVISION OF LOCKHEED AIRCRAFT CORPORATION

(1)

(14) REPORT NO. LR-28460

DATE 7-14-78

MODEL Independent Research

COPY NO. 40

(6) TITLE

SUMMARY OF 1977 INDEPENDENT RESEARCH ON USER ORIENTED REMOTE COMPUTING,

(11) 14 Jul 78 (12) 54 p.

REFERENCE 21-3715 5890

CONTRACT NUMBER(S)

PREPARED BY R. W. Lingard

(10) R. W. Lingard, Sci. Comp. App. Spec., Sr.
Simulation Analysis Programming

PREPARED BY [redacted]

D. Saiki, Sci. Comp. App. Programmer
Simulation Analysis Programming

APPROVED BY H P Weinberger

H. P. Weinberger, Group Engineer
Simulation Analysis Programming

APPROVED BY J. D. Little

J. D. Little, Department Manager
Scientific Analytical Programming

APPROVED BY R B Ferry

R. B. Ferry, Manager
Scientific Computing Division

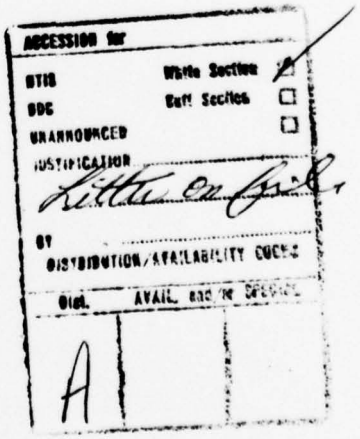
REVISIONS

REV. NO.	DATE	REV. BY	PAGES AFFECTED	REMARKS

FORM 402-2

FORWARD

This document is a report of the third year's accomplishments on the Calac independent research task entitled, "User Oriented Remote Computing" (project number 7701150). The authors are indebted to Thomas R. Jones for the many constructive suggestions he made, and especially to Howard Weinberger for his constant interest and advice during the course of this task.



ABSTRACT

The usefulness of computers in solving scientific problems is a function of the ease with which users can communicate with existing hardware and software. This research is aimed at improving such man-computer communication. Specifically, a computer system has been designed and partially implemented which will provide a software interface between users, possibly inexperienced in computer processing techniques, and available programs and analysis systems.

This system, denoted ASSIST (A Scientific Software Interface System for Terminal users), aids users in accessing and utilizing existing applications software from remote terminals. The system provides three basic functions. It helps users find programs relevant to their problems; it assists them in preparing required input data; and it aids in the actual submittal of programs and data for computer processing. In addition, the system monitors usage of the facilities to ensure the efficient and proper use of available computer resources.

The basic approach has been to design a language which programmers can use to describe program characteristics (function, input format, submittal requirements, etc.). These descriptions can then be interactively interpreted by ASSIST to aid individuals wishing to use any available program. Thus, the user has a helpful interface with which he can converse while he is trying to find, prepare input for, or submit a program.

In addition to improving ASSIST, a major portion of the current research has been directed at determining the proper hardware/software/firmware configuration to support it and other user oriented interactive capabilities.

TABLE OF CONTENTS

	Page
FORWARD	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	iv
1.0 ASSIST IMPROVEMENTS	1-1
1.1 PROGRAM SELECTION	1-1
1.2 INPUT PREPARATION	1-2
1.3 PROGRAM SUBMITTAL	1-4
1.3.1 NESTED MACRO CALLS	1-6
1.3.2 CHARACTER STRING MANIPULATION	1-6
1.3.3 ARITHMETIC CAPABILITIES	1-7
1.3.4 PROGRAMMER'S AIDS	1-7
1.3.5 RESPONSE IMPROVEMENT	1-7
1.3.6 OTHER FEATURES UNDER DEVELOPMENT	1-8
1.4 USAGE MONITORING AND CONTROL	1-10
1.4.1 MONITORING RUNPROG USAGE	1-10
1.4.2 ACTIVITY REPORTING	1-12
2.0 EVALUATION OF COMPUTING ENVIRONMENTS	2-1
2.1 EVALUATION CRITERIA	2-2
2.1.1 CAPABILITIES	2-2
2.1.2 EASE OF USE	2-12
2.1.3 EFFICIENCY	2-13
2.1.4 AVAILABILITY	2-14
2.1.5 EXPANDABILITY	2-14
2.1.6 RESPONSIVENESS	2-14
2.2 EVALUATION OF ENVIRONMENTS	2-15
2.2.1 CENTRALIZED ENVIRONMENTS	2-15
2.2.2 HIERARCHICALLY DECENTRALIZED ENVIRONMENTS	2-18
2.2.3 NON HIERARCHICALLY CONNECTED ENVIRONMENTS	2-22
2.2.4 UNCONNECTED ENVIRONMENTS	2-22
3.0 CONCLUSIONS AND RECOMMENDATIONS	3-1
REFERENCES	R-1

INTRODUCTION

Technological advances in hardware have made computers practical and economical tools for ever increasing numbers of users. More and more people with less and less programming experience will be using computers in the years to come. No longer can systems be designed without consideration of these ultimate users. The effectiveness of future systems will be measured by the ease which man can communicate with them.

Although a great quantity of problem solving software is available today, most is usable only by those with backgrounds in computing. Non-programming users at Calac traditionally depended upon professional programmers as their interface with existing software. In the computing environment prior to 1975, it was the professional programmer who directly accessed both the computer and the library of existing programs. A non-programming user, with a problem to solve, would explain it to some programmer who would perform the necessary tasks to prepare a computer acceptable form of the problem (i.e., put together program and data with required system control information). The programmer would then accomplish the actual computer processing and return the results to the user. This mode of operation had obvious inefficiencies for many kinds of problem solving. There were often delays in processing and errors due to misunderstandings. With more and more people requiring more and more computer processing there was clearly a need to put them in closer contact with the computer.

As a consequence of this conclusion, a Direct Computer Access System, DCAS, was established which logically extended the computer to allow access to it through remote terminals. Initially DCAS existed only as a subset of IBM's Time Sharing Option (TSO). Although this gave a user direct access to the computer, it did not improve his access to the library of programs. This research has dealt primarily with the task of extending DCAS to improve the user's ability to directly utilize this

collection of existing applications software. In effect, the goal has been to automate the interface function previously provided by the programmer. In order to accomplish this the programmer must be provided with some means for transferring his "knowledge"(or information in his keeping) regarding the use of specific analysis software to DCAS. ASSIST is that augmentation of TSO which gives DCAS this capability.

ASSIST has been designed to bring together information regarding existing application programs and potential users in an interactive environment (see Figure 1). For each program to be made available through ASSIST, certain descriptive information is provided by a programmer. This information includes a general program description, a complete and precise input specification, and certain job control information necessary for running the program on the computer. The user can then interact with various components of ASSIST, which have access to this programmer supplied information, for solving some problem. If he needs information regarding the availability of certain software, he can access the Program Selection component. This will tell him about existing programs in a particular category he selects. Once he knows which program to run he may elect to access the Input Preparation component to assist him in preparing his data. He may ask to be prompted for every quantity needed or merely to have his data checked for completeness and, to some degree, correctness. Finally the user can access the Program Submittal component which will automatically create all necessary job control information and submit the specified program and data for execution.

During 1975 a preliminary design of ASSIST was completed. The component of the system which assists users in submitting programs was developed and put into controlled use for testing. This work is described in the report, "Engineer Oriented Remote Computing" (LR 27518). In 1976 an initial production version of ASSIST, containing extensive users aids for program submittal, was completed and put into use. Other aids were implemented and several more were designed including software to help

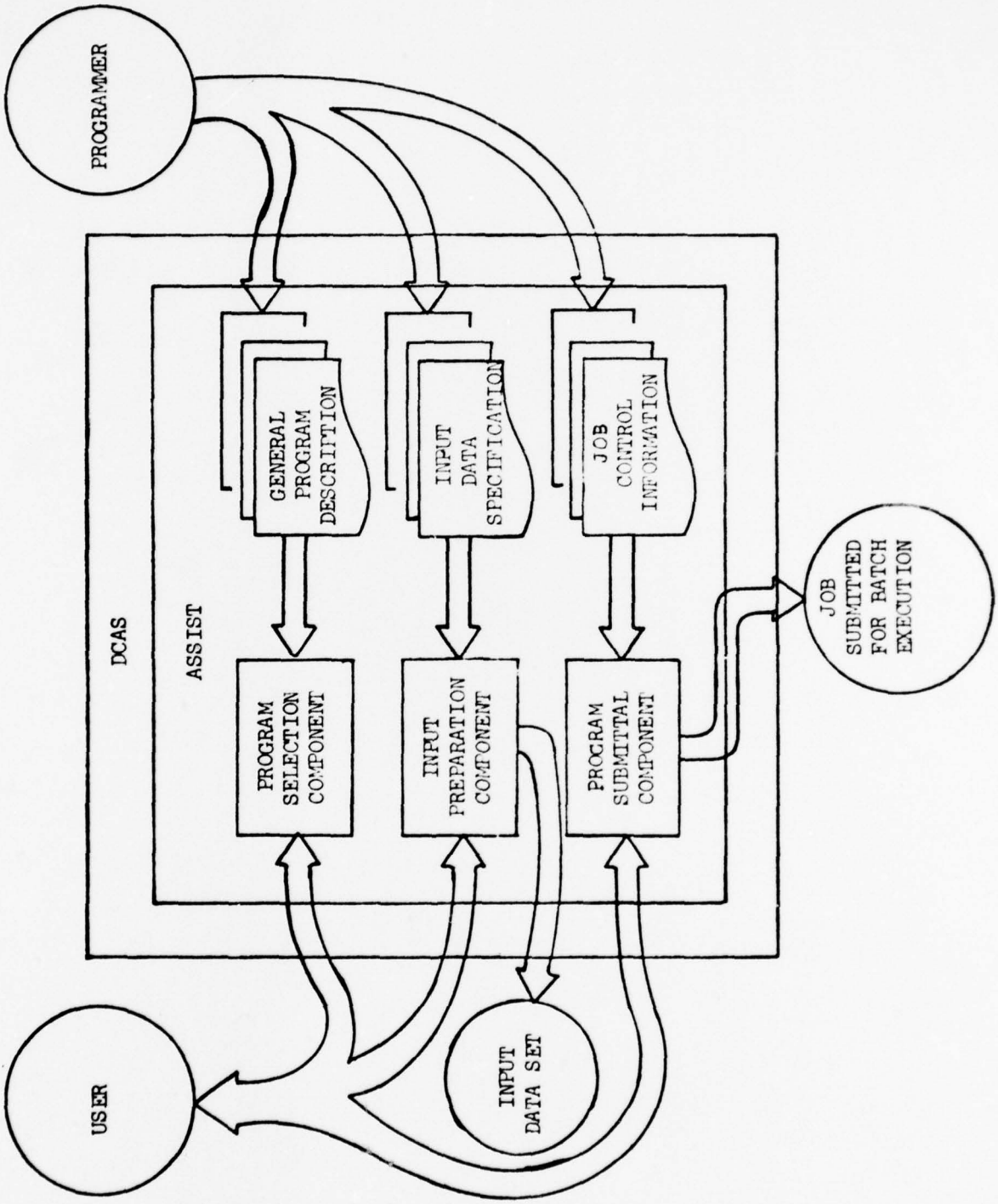


Figure 1. ASSIST

LOCKHEED

users in preparing program input data. A detailed account of the 1976 progress on the various components of ASSIST can be found in the report, "Summary of 1976 Independent Research on Engineer Oriented Remote Computing" (LR 28005).

The specific objectives for 1977 were two-fold. The first was to improve the overall capabilities and performance characteristics of ASSIST, and the second was to find a more suitable computing environment for hosting ASSIST and other interactive capabilities. In particular, the feasibility of a distributed computing approach was evaluated. This report is divided into two major sections, the first being a description of ASSIST improvements, and the second an evaluation of various computing environments.

SECTION 1.0

ASSIST IMPROVEMENTS

The basic design of ASSIST was accomplished in 1975 and has remained relatively unchanged since that time. Implementation of the various components of the design have proceeded according to greatest need. Of the four major components of the system,

- o program selection
- o input preparation
- o program submittal
- o usage monitoring and control

only program submittal has achieved full operational status. The others exist in various stages of completeness. This section gives the present status of each component and describes in detail the accomplishments of 1977.

1.1 PROGRAM SELECTION

The purpose of this component of ASSIST is to provide information to the user regarding available applications software. The nature of this information will be such that he may determine which, if any, available programs might be applicable to a given problem. Program titles, abstracts, development and revision dates, names of responsible programmers, and program identification (Reference File) numbers are examples of the information to be provided. Additionally, this information will be made conveniently accessible from a terminal. Specifically, a user will be able to give a keyword and a list of program titles will be searched for the occurrence of the word given. The program titles corresponding to matched keywords are returned to the user. He may then list the abstract and other information desired for selected programs. One of the primary advantages of such a capability is that it provides an effective means for disseminating information regarding available software

throughout a large community of users.

This component was developed by creating an on-line data set with an entry for each of the PSI (Program Setup Instruction) macros accessible through the program submittal component. In some cases there may be more than one macro for a given available program, but there is always at least one. The entry for a given macro contains its name (a one to eight character identification), a title, the name of the responsible programmer, and the program reference file (RF) number of the program accessed by the macro. This RF number is a key into an existing data base of program description information, the Program Reference File. This data base is maintained by the Scientific Computing Division and contains the remainder of pertinent information for applications programs.

During 1977 the feasibility of linking the Program Reference File directly to the PSI macro data base was studied. This would have allowed interactive users direct access to Reference File information. Also studied was the feasibility of extending the data base searching capability to include Boolean combinations of keywords. Studies of user needs, however, cast doubts as to the importance of these capabilities in relation to the computer resources that would be required to support them. Consequently, no improvements have been made to the program selection component of ASSIST, and none is contemplated until a more definitive assessment of user needs can be obtained.

1.2 INPUT PREPARATION

The purpose of this component of ASSIST is to aid the user in preparing input data for a program he has chosen to run. It will do three things for the user. It will tell him what input quantities are required for a given program; it will enable him to provide those values in a convenient manner (without requiring that he know the specific data formats required by the program); and it will check the data he provides both for

completeness and correctness. Furthermore, this component can be used interactively, prompting the user when necessary and allowing him to correct errors as they are discovered. The effect of this component is to put a user's guide to a program on-line and in such a way that the user can converse with it.

The approach taken for this component is basically the same as that used for program submittal (See Section 1.3). In particular a language has been developed which allows programmers to describe program input requirements. Actually this language is just an extension of the PSI macro language since the fundamental requirements of this component are identical with those of program submittal. Namely, it must interact with a user, providing some information and obtaining other information, and based on that, construct a data set. In the case of program submittal, the data set built is job stream input while in the case of input preparation, it is a data set for input to some program. These differences in no way affect the logical operation of the component of ASSIST which interprets programmer written descriptions and interacts with users. All that is required, therefore, to be able to provide assistance to the user in input generation is to add certain constructs to the existing macro language. In particular, constructs are needed which allow the description of required input parameters, including types and ranges of acceptable values, and the format in which the program expects them to be given.

With such an expanded macro language, a programmer can describe the input requirements for any program in such a way that data prepared for that program can be automatically checked for completeness and correctness, and, if desired, the input can be prepared in an interactive mode. In the latter case, information will be requested of the user through an input description (ID) macro written by a programmer. Normally, this request will be a list of input parameters for which the user must supply values. Additional messages can be displayed to the user at the discretion of the programmer writing the ID macro. The user can ask for a description

of any parameter requested and the values he supplies will be checked for proper type (e.g., character, integer, etc.) and limits (e.g., $0<X<10$) according to information specified in the ID macro. The user will be immediately notified of any errors and allowed to correct them. The input values will then be formatted as required by the program. When this component is used just to check a prepared data set, the data set must be properly formatted. In this case, a list of all discovered errors will be returned to the user. The use of this component could result in significant savings of computer resources by helping users to prepare program input data which are correct the first time.

A software specification for the extended macro language was developed during 1976. The design, coding and testing of the required software modules began in 1976 and continued into 1977. These activities were suspended, however, when results of the computing environments study (see Section 2) indicated that a major hardware/software configuration change might prove beneficial. It was decided to defer further development efforts on the input preparation component until the future computing environment for ASSIST was established.

1.3 PROGRAM SUBMITTAL

The purpose of this component of ASSIST is to simplify the task of program submittal by automatically generating the necessary job control information required by the operating system for program execution. The fundamental concept is that non-programming users should not be required to learn the details of interfacing with the operating system in order to run jobs on the computer. The users should be able to communicate their needs in terms meaningful to them not in the language of the operating system. For example, a user desiring plot output should merely have to say, "PLOTS" or respond affirmatively to the question, "Do you want the output plotted?" rather than have to know how to appropriately modify the DD (Data Definition) statement of the associated plot file. Such capabil-

ities could be of great benefit to the experienced programmer as well, for even with a knowledge of JCL (Job Control Language), it might be far simpler to allow ASSIST to automatically generate necessary control information. Certainly there is much less chance of error for either the experienced or inexperienced user when the program setup and control information are produced automatically.

The approach taken in this research for assisting users in program submittal has been to design a language, and interpreter for it, which can be used by programmers for expressing the information necessary for running programs on the computer. This language, known as the PSI (Program Setup Instructions) Macro language, is an augmented job control language (JCL) which allows construction of generalized sets of JCL for the IBM operating system. The interpreter acts as a preprocessor or macro processor, expanding programs written in this language into complete and valid jobs to be executed on the computer.

In a typical case, a programmer who is familiar with a particular program and the JCL required to run it will develop a generalized set of job control instructions called a PSI macro. This PSI macro will then be placed in an on-line library and, hence, will be available to all users through the PSI macro processor known as RUNPROG. Once a PSI macro has been so created for a given program, users can run that program by accessing RUNPROG, without regard to any JCL concerns. Furthermore, changes that may be required in the JCL due to program modifications, system changes, or operational considerations can be usually made in the single version of the generalized JCL in the PSI macro library without requiring any change on the part of the users. In cases where programming changes were made, all users will automatically get access to the latest version of the program. Thus, this component of ASSIST can help not only the user in program submittal, but the programmer in program maintenance as well,

During the first year (1975) of this research effort, the origi-

nal design of the program submittal component (RUNPROG) was developed and a preliminary version with limited capabilities was put into production use. In 1976, the capabilities of this production version were expanded in accordance with the original specifications. The initial design of RUNPROG underwent a minor revision during 1977 and several new features were added to the production configuration along with capabilities which were part of the original design. A general description of the modifications and enhancements made during the third year of this research project is given below. A complete description of the current capabilities of RUNPROG can be found in the"Programmer's Guide to ASSIST".

1.3.1 Nested Macro Calls

The capability to invoke one PSI macro from within another macro was added to the production version. Part of the original design, this feature is particularly useful because it allows different PSI macros to access (i.e., 'CALL') a common set of augmented JCL statements (i.e., a PSI macro), thereby eliminating the need for including these statements in each individual macro.

1.3.2 Character String Manipulation

Three new functions not in the original design were added to greatly enhance the character string handling facilities of the macro language. The capabilities to:

1) concatenate character strings (CONCAT),
2) search a given string for a particular configuration or substring (INDEX) and
3) determine the current length of a character string (LENGTH)

have been implemented in the production version. These new functions are very similar to their PL/I counterparts and along with the substring function (SUBSTR) give the user a full range of character string manipulation capabilities.

1.3.3 Arithmetic Capabilities

Due to core limitations, the ability to evaluate arbitrary expressions could not be implemented in the production version. In order to compensate for this lack of arithmetic capability, three new functions have been provided: ADD, SUB and MULT. These functions allow the user to perform the arithmetic operations of:

1) addition,
2) subtraction and
3) multiplication,

respectively and are helpful in partially overcoming one of the most severe limitations of the macro language.

1.3.4 Programmer's Aids

Several utility capabilities have also been added to RUNPROG. The ABEND statement produces a symbol table dump of all macro variables and their current values and is helpful as a diagnostic tool in debugging PSI macros.

The DATE and TIME functions return the current date and time in the following formats:

DATE: MM-DD-YY where MM - month
DD - day
YY - year

TIME: hh:mm:ss where hh - hour
mm - minute
ss - second

1.3.5 Response Improvement

The original implementation of RUNPROG as an executable load module called from within a command clist has been discarded in order to

decrease the elapsed time spent within the RUNPROG processor. Allocation and freeing of data sets from TSO is a slow and costly process and RUNPROG requires several data sets to be allocated during execution. Alternate approaches which could remedy this problem included allocating data sets when a user first logs on and doing this allocation from within the processor itself. Both of these methods were studied and it was decided that a combination of these methods would be used. This change allowed the conversion of RUNPROG from a command clist to a command procedure and eliminated the time necessary to allocate data sets from TSO thereby greatly reducing elapsed time spent executing RUNPROG.

1.3.6 Other Features Under Development

In conjunction with the usage monitoring and control component, the capability to monitor the use of RUNPROG is under development. This function will collect data regarding the use of the program submittal component (RUNPROG) and should provide statistical information for evaluating RUNPROG.

Several capabilities included in the original design at RUNPROG are also under development. These include:

1) expanding the number of allowable relational operators,
2) extending the compound conditional statement and
3) allowing TSO data sets to be copied into the control file being built.

At present, only the '=' (equal) and '¬ =' (not equal) relational operators are allowed in the evaluation of a logical expression. This set of allowable relational operators will be expanded so that other conditions may be tested (e.g., '>' (greater than) and '<' (less than).

The complete compound conditional statement has the following format:

IF condition THEN statement

ELSE statement

The capability to execute a statement or group of statements if the tested condition is false (i.e., an ELSE clause) will be developed to allow use of the complete format of the conditional statement.

During 1977, the ability to copy a TSO data set into the control file being built was studied and developed. However, during the testing phase several problems occurred concerning the use of system routines and implementation was delayed until further tests could be made. This feature is still in the test phase.

The concept of being able to execute the RUNPROG processor in the batch mode when TSO is not active has been studied and partially developed. Designed originally for a terminal-oriented, time-shared environment, this enhancement would allow the user to access the PSI macro library and execute the RUNPROG processor even when TSO is not available.

Other enhancements to RUNPROG which will be studied or developed include off-loading inactive macros from the PSI macro library, consolidation of the current error messages into a comprehensive scheme and a method of copying user-supplied input variables and their values to the control file being built.

1.4 USAGE MONITORING AND CONTROL

The purpose of this component of ASSIST is to ensure the efficient usage of available computer resources by developing adequate system controls and through the monitoring of user activity. Since extensive computer software and hardware resources have been made available to non-programming users, there is a need to prevent inadvertent misuse due to lack of computer experience. As a minimum, sufficient information must be collected in order to determine whether the resources are being efficiently utilized. The information so collected, since it will reflect user activity, will also be valuable for guiding efforts to improve the efficiency of ASSIST itself.

The accomplishments toward prevention of accidental misuse of resources were designed and implemented within the Program Submittal and Input Preparation components of ASSIST. By their very nature these components eliminate many sources of user errors. The Program Submittal component automatically determines many required parameters, and both components have capabilities for checking the correctness of user supplied values. In the case of program submittal, the computer resources requested (e.g. core, time, lines of output, etc.) can be controlled, and, in the case of input preparation, the execution of runs with erroneous data can be prevented.

1.4.1 Monitoring RUNPROG Usage

Beyond these capabilities, the primary method for ensuring the efficient use of resources has been through the collection and analysis of data relating to user activity. Much of the relevant data is available for terminal sessions just as it is for normal batch work through the standard accounting procedures. In 1976, an experimental module was designed and developed to collect information regarding jobs processed by the Program Submittal component (i.e. RUNPROG). Because the addition of this function

would have only aggravated the already poor response time, it was decided that implementation of the user monitoring function be delayed.

During the last quarter of 1977, the Program Submittal component underwent a minor design revision which improved response time significantly (see Section 1.3.5). This improvement allowed the User Monitoring component to be implemented. However, it also required a change in the design of the monitoring of user activity.

In order to prevent contamination when collecting information about user activity, two new functions were identified and incorporated into the User Monitoring module. One function prevents simultaneous update of the same record by several users and the other prevents a user from interrupting the update process.

When two or more users are allowed access to the same data base, there is always the possibility that they will attempt to update the same record at the same time, the result being that only one will be successful. Recognition of this problem led to the first function which queues users, allows only one user at a time to update the data base and then dequeues users.

Initial implementation of the first function demonstrated the need for the second. If a user is queued and attempts an attention interrupt, he abnormally terminates from the processor but does not vacate his position in the queue. Because the user will never perform the update and be removed from the queue (i.e. dequeued), all users behind him will remain waiting in the queue until it is forcibly emptied (i.e., an IPL occurs). The second function disables the attention interrupt handler when a user enters the queue and enables it when the user exits the queue thereby avoiding a possible bottleneck in the system.

Testing of these new functions is expected to be completed in

early 1978; implementation is scheduled for the second quarter.

1.4.2 Activity Reporting

A collection of data is meaningless unless it is presented in such a way that useful information can be derived from it, thus the need for reports. A series of MARK IV reports are being designed to provide statistical information to aid in the evaluation of the use of ASSIST and to provide summary information to aid management decisions regarding directions of future growth for the system.

SECTION 2

EVALUATION OF COMPUTING ENVIRONMENTS

It has been a major premise of this research that greater productivity among engineers and other computer users can be achieved by providing capabilities enabling these individuals to directly access the existing computing hardware and software. It has not been a purpose to determine the validity of this premise although it should be pointed out that ample supporting evidence does exist. A study by Integrated Systems Support, Inc. on "A Production Environment Evaluation of Interactive Programming" (16) demonstrated both an increase in user productivity and a decrease in overall computing costs by providing users with direct interactive access to the computer through remote terminals. Furthermore, questionnaires distributed to users at Calac indicated a feeling of greater productivity when direct computer access capabilities were available.

The extent to which direct access capabilities are provided, the specific functions to be performed, and the means of doing it have been the subject of this research task and other related efforts over the past several years. The result of these efforts has been the development of the computer based system, DCAS (Direct Computer Access System). In its present implementation DCAS exists in a centralized environment. Specifically, the system functions are imbedded within IBM's TSO (Time Sharing Option) portion of the operating system OS/MVT on a 360/91 computer.

Unfortunately, the present system does not adequately satisfy current needs and has very little growth potential. It's capabilities are far too limited, it is difficult to use, and it is unresponsive to the user. In addition, the present hardware is unreliable and the system software (i.e. TSO) is highly inefficient. Even with all these drawbacks users generally agree that DCAS is far superior to past methods of operation where computer processing was done in a batch mode by (or with the

assistance of) professional programmers.

Since it seemed to be only the particular implementation and not the basic system design that limited the effectiveness of DCAS, this year's research was directed at investigating the feasibility of implementing DCAS within other hardware/software/firmware environments. In particular, various forms of distributed computing environments were studied and evaluated. This section is devoted to that evaluation.

2.1 EVALUATION CRITERIA

Six areas were selected as a basis for judging system effectiveness. Specifically, the environments were evaluated with regard to how well they could support systems which would provide the required capabilities, be easy to use, efficiently utilize computer resources, be reliable, offer growth potential, and be responsive to the users. These criteria are more fully discussed in the following sections.

2.1.1 Capabilities

It is probably of primary importance that the specific capabilities of the system be precisely established. This is a particularly difficult task since needs are influenced by available capabilities. It is usually difficult to express needs independently from existing tools. When asked what their needs are in order to be able to do computing more effectively, users respond by saying such things as,

"We need a bigger computer." (or worse, "We need a CDC 7600.")

"We need remote terminals." (or worse, "We need some HP 2641A's")

"We need a time sharing system." (or worse, "We need TSO.")

When expressed in functional terms the need for a larger computer becomes a need for the means to solve larger problems or for solving them faster. Likewise, the need for remote terminals might become the need to be able to conveniently communicate with the computer from remote locations.

Finally, requesting a timesharing system probably indicates the need for some interactive computing capabilities.

In order to find the best means for providing direct access capabilities, needs must be expressed in a functional, tool-independent manner. It must also be understood that user needs will continually grow (and even change). Increased capabilities create new needs and obsolete others. Whatever might be determined to be today's needs will probably be out-of-date in the not-to-distant future.

With these thoughts in mind the following capabilities have been established as the set required to satisfy user needs for computer processing at the present. These capabilities stated as functional requirements, are based upon user provided information and two years of experience with the present implementation of DCAS. The functions can be divided into nine areas.

1) data set creation and manipulation
2) computer program development
3) batch program submittal
4) interactive program execution
5) command set processing
6) user communication
7) document preparation
8) system management
9) system access

The functions can further be divided into essential and desired capabilities. Those classified as essential must be realizable in order for a computing environment to be acceptable. The environment will then be rated according to how well it is suited for providing essential capabilities and how many desired features it is capable of supporting. In the following sections the functional requirements are described. Essential functions are noted with an asterisk (*).

2.1.1.1 Data Set Creation and Manipulation

This is a fundamental capability which must be provided to the user. He must be able to create, identify, and store sets of data. Furthermore, he must be able to retrieve, modify, and display these data sets.

A data set is defined here to be any collection of data. It could be numeric, textual (including lower case letters and special symbols), or an actual computer program. The data sets themselves consist of lines (records) which may be of any size. The lines (records) of a data set may be all of one size or of varying sizes. A given user may create and keep several data sets. This collection will be referred to as his library of data sets.

The specific data set handling functions are:

* A) create - form and identify a data set by entering information into the system.
  B) reidentify - give a new name or an alias to an existing data set
* C) store - retain a data set within the system identified by its name
* D) retrieve - fetch a data set with a specified name
* E) list - display the contents of a data set or a portion thereof
* F) delete - remove a data set from the system
* G) copy - duplicate a data set or a portion of it optionally rearranging and/or eliminating certain columns (fields) of lines (records) in the copy
  H) describe-as - associate a textual description with a data set
  I) protect - prevent access to and/or modification of and/or deletion of a data set except by authorized users

J) modify - change contents of a data set as follows:

* 1) insert - add one or more lines (records) to a data set at some location

2) replace - substitute one or more lines (records) with another line (record) or set of lines (records)

* 3) delete - remove one or more lines (records) from a data set

4) move - transfer one or more lines (records) from one place to another within a data set

* 5) copy - duplicate one or more lines (records) one or more times from the current or other data set into the current data set at some location

6) search-for - find the first occurrence, first n occurrences, or all occurrences of a string within a range of lines of a data set, optionally limiting the search to certain columns (fields) within the lines (records)

7) change - alter one or more lines (records) of a data set as follows:

a) insert - place one or more additional characters at some point within a line (record)

b) delete - remove one or more existing characters from a line (record)

* c) alter - replace one string with another string in one or more lines (records) optionally limiting replacements to certain columns (fields) within the lines (records)

K) undo - restore data set to the form it had just prior to execution of the last command or last n commands

* L) list-catalogue - display the names, and optionally descriptions, of all data sets in a user's library

M) compare - test two data sets, or portions thereof, to see if they are identical and list any differences, optionally specifying the columns (fields) in the lines (records) to be compared

N) shift - move one or more lines (records), or portions thereof a specified number of columns (positions) right or left

O) sort - rearrange lines (records) or a data set into either ascending or descending order based upon a selected range of columns (field) in the lines (records)

P) check - test a data set for compliance with some established format

2.1.1.2 Program Development

Users must be able to interactively create and check out computer programs. As a minimum, these capabilities must be available for the FORTRAN programming language. It is also highly desirable that these capabilities be available for PL/I, Assembler Language, and APL. The specific functions needed are as follows:

A) scan - syntactically check a statement or set of statements for compliance with the rules of the particular language

B) compile - generate object code from a given source statement program displaying any compilation errors to the user

C) assemble - cause an Assembler Language program to be assembled (i.e., generate object code)

D) link - generate a load module from a given object module

E) debug - test a program for proper execution and locate programming errors as follows:

1) execute - cause execution of the program to begin at a specified statement and optionally specify where execution is to stop

2) interrupt - cause execution to cease immediately

3) list-variables - display the current value(s) of program variable(s)

4) change-value - modify the value of a variable or set of variables

5) list - display a statement or set of statements

6) modify - change (including all subfunctions described as part of the "modify" function under "Data Set Creation and Manipulation") the contents of the program
7) set-stop - designate a statement in the program where execution will stop whenever the statement is reached
8) remove-stop - remove the designation that execution should stop at some statement
9) trace-variables - display the value of a specified variable whenever the value changes
10) trace-flow - display indication of each transfer of control during program execution

2.1.1.3 Batch Processing

Capabilities must be provided to allow users to perform batch processing of computer programs. The specific functions needed are as follows:

* A) submit - cause a program to be submitted for batch execution
* B) list-output - display the output of a job run in batch mode
* C) determine-status - return the current status of a job submitted for batch execution
* D) cancel - cause a job submitted for batch execution to be cancelled
E) scan - syntactically check job control statements
* F) save-output - put output from a job into a data set
G) access-data-set - provide access to a data set from a job executing in a batch mode
H) plot - display plot output

2.1.1.4 Interactive Program Execution

Users need to be able to create and run interactive programs. The specific functions needed are as follows:

* A) run - call up and execute a program interactively
  B) suspend - cause an interactive program to temporarily halt execution
  C) resume - cause a suspended interactive program to resume execution
* D) cancel - terminate execution of an interactive program
  E) access-data-set - provide access to a data set from a program executing interactively
  F) calculate - determine and display the result of a specified numerical calculation
  G) plot - generate and display a plot from data given

2.1.1.5 Command Set Processing

A capability to execute sets of interactive commands as a group must be provided. Ideally, the command language will be sufficiently rich to allow a full programming capability at this level. That is, logical testing, branching, looping, and input/output constructs should be part of the command language. Specific functions needed are the following:

* A) execute - execute a set of interactive commands
  B) suspend - cause the execution of a command set to be temporarily halted
  C) resume - cause the execution of a suspended command set to resume
  D) scan - syntactically check statements of the command language

2.1.1.6 User Communication

It must be possible for those maintaining the system to communicate with users and for users to communicate with each other and the system operator. The following specific functions are required:

* A) send - send a message to a user, a set of users or the system operator
* B) hold-messages - cause messages (except those sent by the operator) to be held in the user's message file rather than be displayed immediately
* C) list-messages - display and delete all messages in the user's message file
  D) associate-message - cause a message to be associated with a data set such that any time the data set is accessed the message is sent to the user accessing it

2.1.1.7 Document Preparation

Capabilities are needed to enable users to create, modify, and maintain textual documents through the system. All functions described in section 2.1.1.1 on "Data Set Creation and Manipulation" must be available for document data sets, and in addition the following functions are required:

A) format - specify which lines are to be formatted before displaying and which are to be displayed as they are by use of the following functions:
   1) set-line-width - cause lines making up paragraphs (to be formatted) to be adjusted so that each line will contain as many words of the text as possible without exceeding the line width specified
   2) start-paragraph - cause a new paragraph to begin at a specified point
   3) center-line - cause line to be centered on page
   4) indent - specify manner of indention for a paragraph

5) suppress-window-lines - prevent the first line of a paragraph from being the last line on a page or the last line of a paragraph from being the first line on a page

B) set-page-depth - cause all pages to have the size specified

C) title - cause a title to be put on every page

D) number-pages - cause all but the first page to be sequentially numbered

E) set-spacing - specify number of blank lines to be inserted between lines of text

2.1.1.8 System Management

Capabilities must be provided to control access to and use of the system including the reporting of resource usage information. It must be possible to specify for each user an identification which will be the name by which the system knows the user. Also, the capability to establish a password for a user which will restrict unauthorized access, and to specify, for each user, a set of valid account numbers for the purpose of charging the computing resources expended. Additionally, a capability to define for each user his limits of access and use of resources is required (i.e., the functions and the types and amounts of resources he is allowed to use). Finally, it must be possible to keep any other information about users as required (e.g., this may include his organization identification, address, phone number and the like). The specific functions required to maintain all this user information and monitor resource utilization are as follows:

* A) add-user - enter a new user and his information set to the system

* B) change-information - modify the user information

* C) delete-user - remove a user from the set of authorized users

* D) list-users - generate a list of all users or a subset thereof, optionally listing any of the user associated information

* E) cancel-user - terminate the session of an active user

* F) charge - determine and optionally display the computer resources expended by a user for the current session or over some period of time
* G) report-resource-usage - generate a report containing all computer resource usage by all or a subset of all users over some period of time
  H) report-function-usage - generate a report containing statistics on usage of all system functions over some period of time (including response)
* I) send-broadcast-message - enter a message into a broadcast data set which is displayed to all users each time they log on to the system
* J) delete-broadcast-message - remove a message from the broadcast data set
  K) monitor-user-activity - cause all terminal input and output of a user to be displayed or printed elsewhere

2.1.1.9 System Access

The means to identify oneself to the system in order to gain access to its capabilities and to specify to which account the expenditure of resources is to be charged must be provided. The following functions are required:

* A) logon - gain access to the system by specifying user identification, possibly a password, and the account number to which computing resource expenditures are to be charged
  B) change-account - specify that all subsequent work is to be charged to the new account number specified
* C) logoff - exit from the system

2.1.2 Ease of Use

A system can provide all essential capabilities and still be difficult to use. An important feature of any system, therefore, is its ease of use. Ease of use encompasses many varied concepts including the following:

A) Learning - Can the system be easily learned? Can an individual learn how to use it on his own or is formal training required? How long does it take to become proficient?

B) Prompting - Is the user prompted when he fails to supply required information?

C) Tutoring - Can the user obtain help from the system when he doesn't know how to do something?

D) Use of Abbreviations - Can the user abbreviate commands? Can he abbreviate other information such as his data set names?

E) Editing Data Sets - Are there features to facilitate the modification of data sets?

1) Can lines of a data set be referred to by line numbers (whether or not the numbers are part of the data set itself)?
2) Can a line or set of lines be modified merely by displaying them, making physical changes in them (including overtyping, inserting, or replacing characters), and re-entering them to the system?
3) Can a block of lines be entered a once?
4) Are the results of changes automatically displayed to the user?
5) Can lines of any length be handled?
6) Can logical tabulation points be established?
7) Are various character sets available?

F) Entering Commands - Can a previously entered command be re-entered, possibly modifying it, without completely retyping it?

G) Use of Physical Devices - Are the devices available easy to use?

H) Viewing Computer Output - Can the user move about within his output?

I) Entering Data - Can data be remotely entered without typing (e.g., from tape)?

J) Locally Storing Data - Can data be stored at the users local site (remote from the central facility) in other than printed form (e.g., on tape)?

K) Transferring Data - Can data be easily transferred to a site outside of the environment of the system?

L) Displaying Information - Can the cursor on a terminal be controlled from an interactive program? Are various character sets available on the terminals? Do terminals have blinking, half bright, inverse video, upside down, 90 degree, etc. display features?

M) Sending Messages - Can messages be sent even when the rest of the system is inoperative?

N) Submitting to Batch - Can batch submittals be made without knowing the job control language of the host computer?

2.1.3 Efficiency

An important consideration of any system is its cost in terms of computer resources utilized. The computing environment must make possible the efficient implementation of the required system functions. In particular, the most used functions must be capable of the highest degree of efficiency in their implementation.

2.1.4 Availability

Availability refers to the degree to which a user has access to the computing capabilities he needs. This encompasses two attributes of the computing environment. The first is its capacity. There must be sufficient computing power and numbers of terminals and other peripheral hardware to serve the needs of the users during times of peak system activity. The second attribute is system reliability. The system must be kept running at or above the 98% level during prime shift hours. Furthermore, the computing environment should be such that it contributes toward implementation of a system which can continue to operate at some level even when portions of the hardware and/or software are not functioning.

2.1.5 Expandability

Another important attribute of any computing environment is its flexibility in terms of size and capabilities. Can the system be easily expanded to handle more users? Can its size be diminished if the work load decreases? Can new capabilities be easily incorporated into the system? The system must be capable of growing or shrinking in a cost effective manner as changes in the work load and needs of the users occur.

2.1.6 Responsiveness

Finally the speed with which the system can act upon user requests must be considered. This is possibly one of the most critical factors contributing toward user satisfaction. Naturally, the time it takes to complete a task depends upon its complexity, but the computing environment must be such that users are not frustrated by unreasonable delays in servicing requests. As a minimum, the system must be capable of responding to trivial or fundamental requests within one second 90% of the time and within two seconds 99% of the time.

2.2 EVALUATION OF COMPUTING ENVIRONMENTS

Four basic types of computing environments will be discussed and evaluated in this section. There are, naturally, many other ways of classifying environments, and all combinations of those to be described are possible. A complete description of computer interconnection structures has been attempted by Anderson and Jensen (1).

A computing environment can be centralized or decentralized. If it is decentralized it can be hierarchically organized or not. If it is non-hierachical, it can be connected or unconnected. Thus, the four classes of environments which will be evaluated are:

A) centralized
B) hierarchically decentralized
C) non hierarchically connected
D) unconnected

2.2.1 Centralized Environments

A centralized environment is one in which a single computer (agent) performs all system functions. This is essentially the present DCAS environment where a number of terminals are connected to a central computer (an IBM 360/91) and operate in a time sharing mode (TSO). The centralized environment is depicted in Figure 2.

Because of the power of the central computer, this environment has the potential to provide almost every desired capability. In fact, time sharing systems exist for most large machines which already provide most of the desired capabilities as specified in section 2.1.1. Thus, very little new software development would be required in such an environment. Although there is no inherent reason that systems implemented in a centralized environment should be difficult to use, experience with one such system, TSO, has shown that ease of use is not necessarily guaranteed in such an environment.

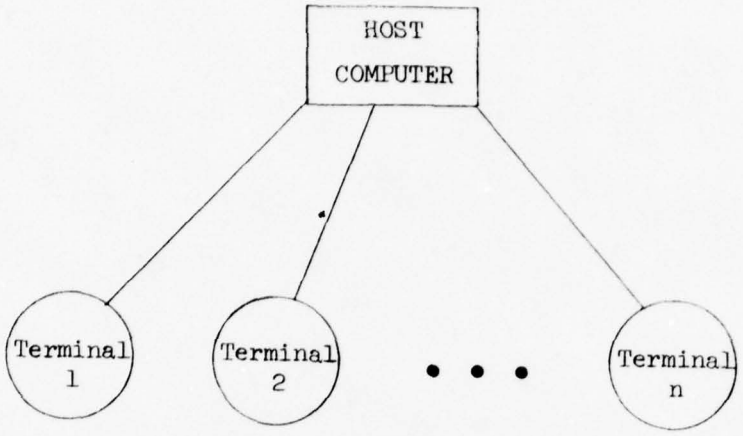


Figure 2. Centralized Environment

The more capabilities a system has the more likely it is to be difficult to use. Hence, there may be tradeoffs between "capabilities" and "ease of use" in the evaluation of a computing environment. In a centralized environment, however, it is clear that these two criteria can be satisfactorily met.

The next evaluation criterion is efficiency. Centralized environments which service many simultaneous users operate, by necessity, in time sharing mode, and time sharing is inherently inefficient. In particular, there is a significant overhead associated with swapping users in and out of main memory and in keeping track of all active users. A significant utilization of computer resources is also required to control the time sharing activity itself. Although computers exist which have been designed to operate efficiently in a time sharing environment, such is not the case with large scale computers of the IBM 360/370 variety.

As far as availability is concerned the centralized environment suffers from the problem that when the host machine is not operating, the entire system is unavailable for use. Thus, system availability is dependent upon the reliability of the central host. Experience with the IBM 360/91 has shown this to be a serious problem.

The expandability or growth potential in a centralized environment is likewise limited by the host machine. Once the capacity of the machine is reached the system cannot grow easily. Also, should a decrease in interactive usage occur, the unused capacity of the central machine might be wasted.

The final criterion is responsiveness. Responsiveness in a centralized environment is primarily a function of the efficiency of the central host. Experience with TSO on the IBM 360/91 has shown that acceptable response is attainable only through allocation of considerable amounts of machine resources.

The centralized environment can be considered a reasonable approach only if efficient, and thereby responsive, hardware and software can be acquired for interactive use. Even then the questions of system availability (reliability) and expandability (growth potential) must be carefully considered.

2.2.2 Hierarchically Decentralized Environments

Decentralization can imply many different concepts. It can suggest that the computing hardware is geographically distributed, that data is distributed among various computers, or that the computing functions are distributed among various pieces of hardware. In this discussion an environment will be considered to be decentralized if the last of these is true even when hardware is not geographically distributed and when all data is centrally controlled. Borrowing from C. V. Ravi (14), an environment is said to be decentralized whenever computing functions (tasks) are distributed among different computing agents (hardware, software, and firmware components). Stretching this point even the so called centralized environment previously discussed is actually decentralized because certain trivial computing functions (such as data entry and simple editing) are performed by the terminals. Certainly in the case of intelligent terminals one would have to call the environment decentralized. For the current discussion an environment will be considered to be decentralized only if significant computing functions are performed by agents other than the central host processor.

A decentralized environment is hierarchically organized if the computing agents are divided into levels, if each agent is connected by a communications link to an agent of the next higher level, and if there is single agent at the top level. In other words, the environment is a tree structure with the branches being lines of communication and where agents at one level control sets of agents at the next lower level. For the purpose of this discussion, an environment will be considered to be hierarachical only if no other lines of communication exist. In particular, communication

lines between agents two or more levels apart or between agents at the same level are not permitted. A simple and typical hierarchically distributed environment is depicted in Figure 3.

Environments of this type are capable of supporting the full range of required capabilities since access to the central host, even if indirect, is always possible. Hierarchically distributed environments can promote ease of use for certain sets of users. They may be able to use simpler, limited systems residing on intermediate computers rather than relying on a complex, all encompassing system residing on the mainframe host. In fact, in such an environment there is the possibility of developing or acquiring highly specialized, easy to use systems for particular classes of users.

Efficiency can be much better in a distributed environment than in the centralized case because much of the processing can be accomplished by less costly hardware located (at least logically if not physically) closer to the user. Certain editing functions, for example, might be accomplished directly by a terminal, compilation of source code might be done on a mini-computer, and small calculations might be handled by a micro-processor within a terminal. The off-loading of these, and possibly other such functions, would greatly reduce the amount and frequency of data transfer between the user and the central computer. This would also reduce the time sharing overhead burden and thereby improve the overall system efficiency or cost effectiveness.

Availability of the system in a distributed environment is significantly improved over the centralized case. It is no longer true that all users are dependent upon a single piece of computing hardware. If significant processing capability has been off-loaded, then at least that processing can continue regardless of the state of the central computer. System availability is a function of the extent to which processing capabilities have been distributed and the redundancy built into the hardware configuration. In a distributed environment it might be reasonable to have an extra mini-computer in case one malfunctions while having and extra large mainframe is usually

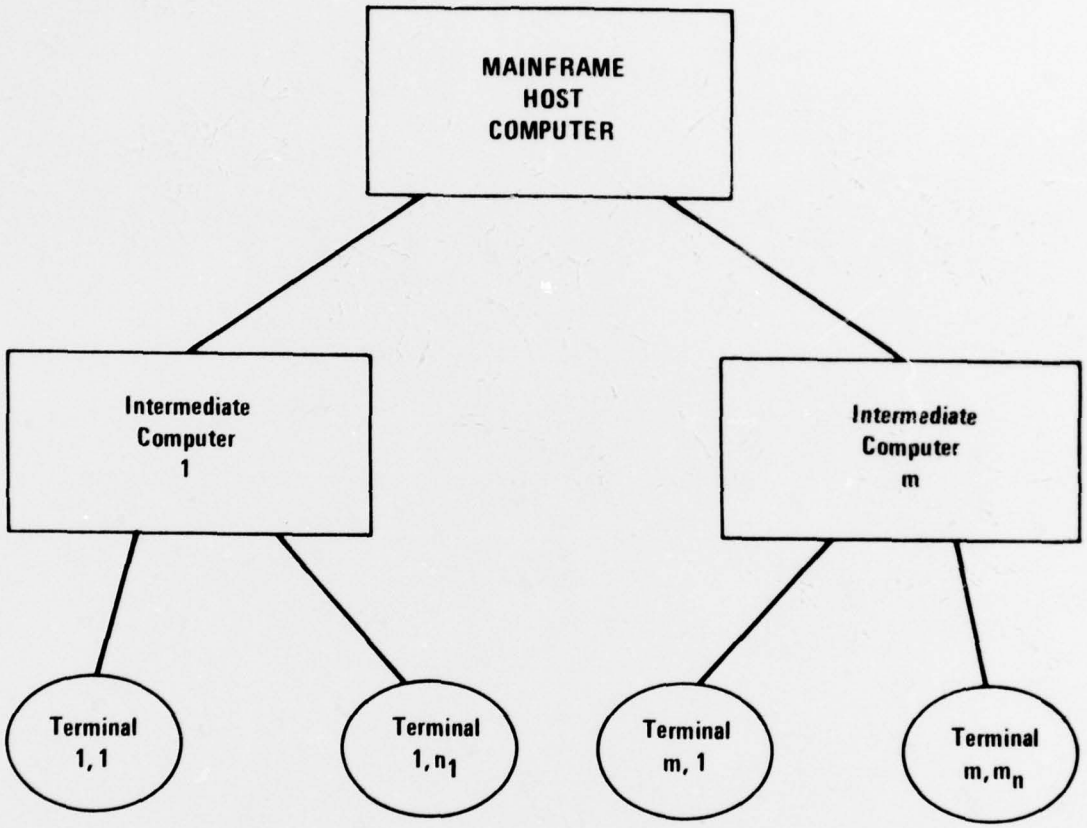


Figure 3. Hierarchically Decentralized Environment

impractical.

The distributed environment is also easily expandable (or contractable) since mini-or micro-computers could be added (or subtracted) as work load conditions changed.

Finally, with regard to responsiveness, the distributed environment has great advantage over the centralized case. For computing functions which have been assumed by intelligent terminals (terminals with micro-processors), the user no longer operates in a time sharing mode. Hence, his response is usually instantaneous. In other cases computing functions may be performed by mini-computers which have been designed for interactive time sharing and are hence more responsive. However, there will undoubtedly be some functions which will take longer to accomplish. For example, any function requiring the transfer of information between the terminal and the mainframe host, or between agents non hierarchically connected, will be less responsive. This fact opens the question of where data should be stored in such an environment. Obviously they should be stored as close as possible to the computing agent which needs access to them. If a set of data is needed at more than one level in the hierarchy, it should probably be stored at the highest level it is needed. While this policy will facilitate central control of data, it might also require great amounts of data transfer. This could have adverse effects on both efficiency and responsiveness.

Clearly, a hierarchically decentralized environment can support a complete interactive computing system. Determining the proper configuration for particular needs, however, is not likely to be an easy matter. The suitability of such an environment and the proper distribution of functions among agents will be largely determined by data transfer requirements.

2.2.3 Non-Hierarchically Connected Environments

This class of environments includes all those which are fully connected but not hierarchically organized. Each computing agent is linked by a communications line to another agent but there are no discernable levels in such an environment. A ring structure like that depicted in Figure 4 is a good example of a non-hierarchically connected environment. The capabilities supportable, ease of use, availability, and expandability characteristics are essentially the same as for hierarchically connected environments. Efficiency and responsiveness on the other hand are dependent upon the specific functions considered. Typically, such an environment gives a user good response for some functions, namely those directly available on the agent to which he is connected. The system responsiveness to other functions depends upon the number of communication paths which must be travelled and the amount of information which must be transmitted. Responsiveness can be improved by adding communications lines, but this will tend to decrease overall system efficiency. Maintaining central control of data, or central control of anything for that matter, in such an environment is difficult since there is no top level computing agent. This would appear to be a serious problem in cases where many users are involved in a single project. If solvable at all, it is likely to adversely affect system efficiency.

2.3.4 Unconnected Environments

This last classification of environments is one in which no lines of communication exist between computing agents. Figure 5 depicts this case. There is nothing inherent in such an environment to limit ease of use, and efficiency, responsiveness, and overall availability are at maximum levels. Since there are no connections between agents, system overhead is minimized and failure of one agent in no way affects the others. Likewise, the system can be easily expanded since adding a new agent can be done without consideration of the rest of the environment. The problem with this environment, of course, is that users are greatly limited in the capabilities available to

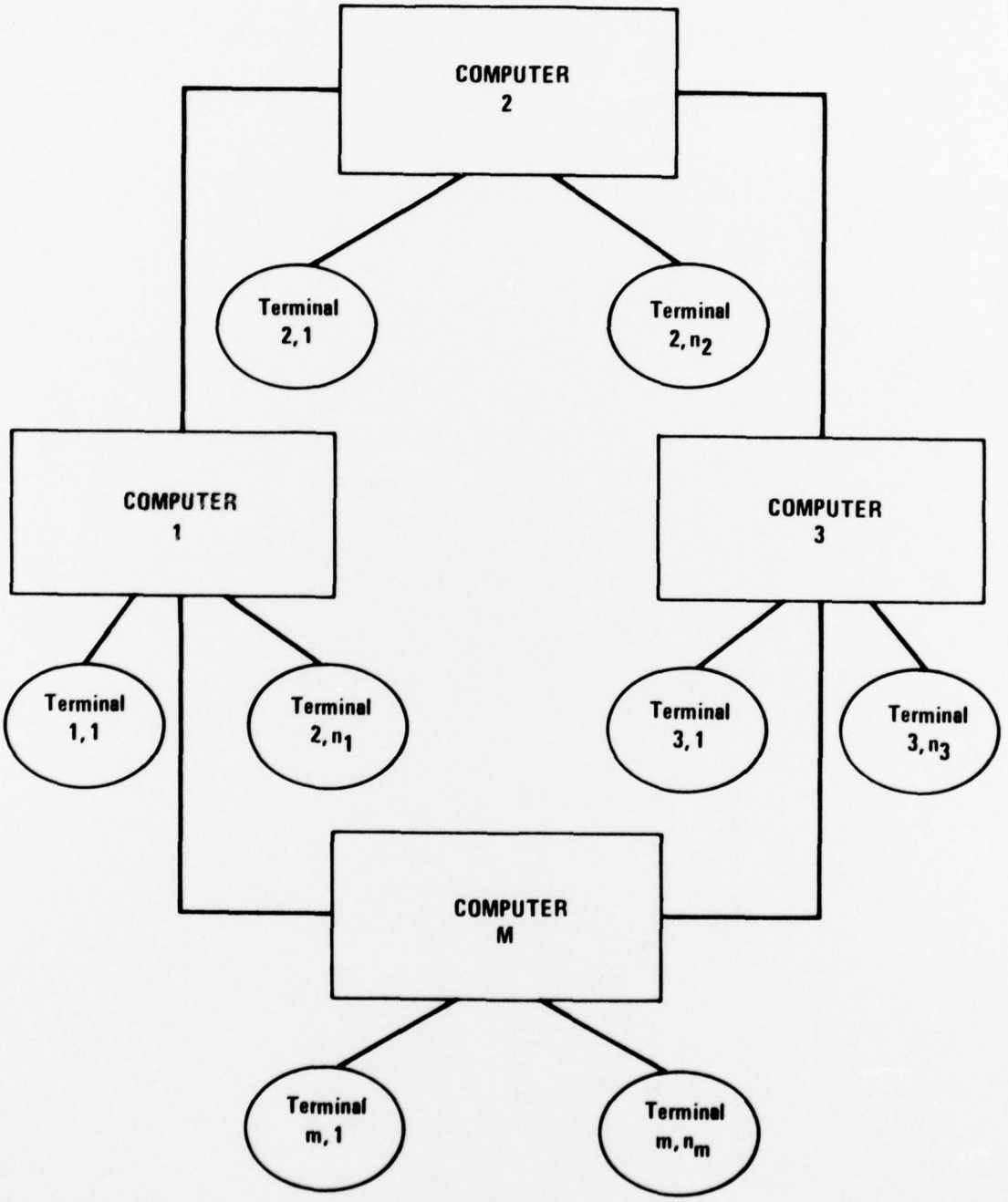


Figure 4. Ring Structure

them. In particular, they can use only those functions available on the computing agents to which they have access. Moreover, managing data or users at a global level is not possible, and communication among users is not supportable.

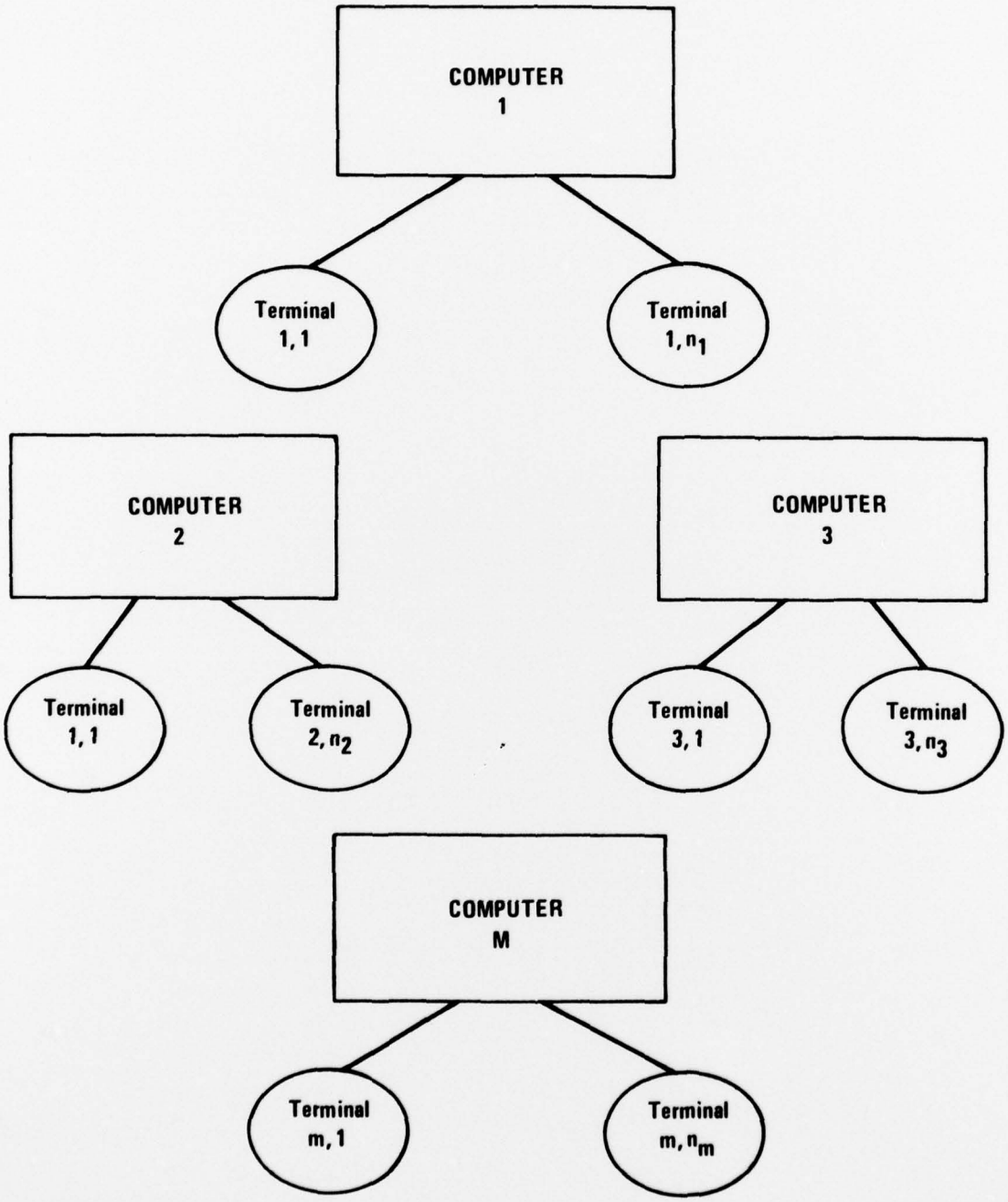


Figure 5. Unconnected Environment

SECTION 3.0

CONCLUSIONS AND RECOMMENDATIONS

The results of the computing environments study are summarized in the table shown in Figure 6. Only the fully connected decentralized environments score at acceptable levels for all evaluation criteria, with the hierarchical organization having a slight edge. Decentralization appears to offer significant potential for both improving service to interactive users and reducing computing costs. Based on this conclusion the following recommendations are made.

	CAPABILITIES	EASE OF USE	EFFICIENCY	AVAILABILITY	EXPANDABILITY	RESPONSIVENESS
CENTRALIZED	+	√	√−	−	−	√−
HIERARCHICALLY DECENTRALIZED	√	√+	√+	+	+	+
NON-HIERARCHICALLY UNCONNECTED	√	√+	√	+	+	√+
UNCONNECTED	−	+	+	+	+	+

+	IDEAL
√+	ABOVE ACCEPTABLE LEVEL
√	ACCEPTABLE
√−	BARELY ACCEPTABLE
−	UNACCEPTABLE

Figure 6

1) Establish a precise and complete specification of user requirements for interactive computing. Although such a specification was produced as part of this research task, it should be reviewed by end user organizations and modified as necessary to ensure that it truly reflects user needs. Furthermore, user requirements need to be ordered in terms of relative importance so tradeoffs can be properly evaluated.

2) Determine the best configuration of computing hardware and software to satisfy the established user needs. This must be a joint effort involving Scientific Computing, Computer Systems, and all end user organizations.

3) Develop a plan for acquisition of any hardware and software necessary to build a prototype of the configuration selected.

4) Prepare a plan for evaluating the effectiveness of the prototype configuration for satisfying users' needs for interactive computing.

REFERENCES

1. Anderson, G. A. and Jensen, E. D., "Computer Interconnection Structures: Taxonomy, Characteristics and Examples," Computing Surveys, Vol. 7, No. 4, Dec. 1975, pp. 197-213.

2. Ashenhurst, R. L., "Centralized or Decentralized Computing - Or Maybe Some of Both?" How to Make Computers Easier to Use, IEEE Compcon 75, Sept. 1975, pp. 59-60.

3. Davis, R. M., "The Systems of the 1980's - A U. S. Perspective," National Bureau of Standards, Nov. 1975.

4. D'Oliveira, C. R., "An Analysis of Computer Decentralization," Massachusetts Inst. of Tech., Cambridge Lab. for Computer Science, Oct. 1977.

5. Doll, D. R., "Relating Networks to Three Kinds of Distributed Function," Data Communications, March 1977, pp. 37-42.

6. Eckhouse, Jr., R. H., Stankovic, J. A., and Van Dam, A., "Issues in Distributed Processing - An Overview of Two Workshops," Computer, Vol. II, No. 1, Jan. 1978, pp. 22-26.

7. Grossberg, M., Wiesen, R. A. and Yntema, D. B., "An Experiment on Problem Solving with Delayed Computer Responses," IEEE Trans. Man and Cybernetics, March 1976, pp. 219-222.

8. Ivie, E. L., "The Programmer's Workbench - A Machine for Software Development," CACM, Vol. 20, No. 10, Oct. 1977, pp. 746-753.

9. Lingard, R. W., "Engineer Oriented Remote Computing," LR 27518, Lockheed-California Company, Burbank, California, Dec. 1975.

10. , "Summary of 1976 Independent Research on Engineering Oriented Remote Computing," LR 28005, Lockheed-California Company, Burbank, California, June 1977.

11. Martin, J., Design of Man-Computer Dialogues, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1973.

12. Melendez, K. J. and Johnson, R. T., "Evaluation of the UNIX Time-Sharing System," Los Aloamos Scientific Lab., New Mexico, Apr. 1977.

13. Miller, L. A. and Thomas, J. C., "Behaviorial Issues in the Use of Interactive Systems," IBM Thomas J. Watson Research Center, Yorktown Heights, New York, Dec. 1976.

14. Ravi, C. V., "The Structure and Characteristics of Distributed Systems," Proc. 2nd International Conference on Software Engineering, Oct. 1976.

15. Reaser, J. M. and Carrow, J. C., "Interactive Programming: Summary of an Evaluation and Some Management Considerations," Integrated Systems Support, Inc., Falls Church, Virginia, March 1975.

16. Reaser, J. M., Priesman, I. and Gill, J. F., "A Production Environment Evaluation of Interactive Programming," Integrated Systems Support, Inc., Falls Church, Virginia, Dec. 1974.

17. Rockhart, J. F., Bullen, C. V. and Leventer, J. S., "Centralization Vs. Decentralization of Information Systems: A Preliminary Model for Decision Making," Massachusetts Inst. of Tech., Sloan School of Management, July 1977.

18. Walton, J. R., "Performance Evaluation of the Lincoln Laboratory Time Sharing System," Massachusetts Inst. of Tech., Lexington Lincoln Lab., May 1976.

19. Weinberg, G. M., The Psychology of Computer Programming, Van Nostrand Reinhold Company, New York, 1971.

ENGINEERING REPORT INITIAL DISTRIBUTION LIST

(SEE EPM 4-07)

REPORT NO. LR 28460

TITLE	MODEL	SECURITY CLASS.	DATE
SUMMARY OF 1977 INDEPENDENT RESEARCH ON USER ORIENTED REMOTE COMPUTING	I.R.	UNCLASSIFIED	7-14-78

APPROVALS

DIVISION ENGINEER: Robert B. Piercy

ORIGINATING ORGANIZATION (TITLE & DEPT. NO.)
Scientific Analytical Programming (80-36)
Scientific Computing Division

COMMERCIAL ENGINEERING (COMMERCIAL ENGINEERING BRANCH REPORTS)

PRODUCT EVALUATION GROUP: Eileen Fuller for L.G. Turner

WO/EWA: 21 CLASS — 3715 WORK ORDER — 5890 E.W.A.

REMARKS

LEGAL BRANCH - PATENT SECTION (STATE ANY RESTRICTIONS): None
PUBLIC DOMAIN

LIMITATION ON ACCESS TO DATA:

UNLESS LIMITATIONS ON SUBSEQUENT RELEASE OF THIS REPORT ARE STATED BELOW, COPIES WILL BE MADE FREELY ACCESSIBLE TO ALL CORPORATION EMPLOYEES. (IF LIMITED, SUBSEQUENT RELEASE TO OTHER ORGANIZATIONS REQUIRES COMPLETION OF FORM 7229.)

LIMITED TO:

REASON:

DATE ON WHICH LIMITATION MAY BE LIFTED:

WOULD IT BE BENEFICIAL TO CALAC TO RELEASE THIS REPORT TO THE PUBLIC VIA NASA/DoD LIBRARIES? ☒ YES ☐ NO
(ANSWER THIS QUESTION FOR INDEPENDENT RESEARCH OR INDEPENDENT DEVELOPMENT FUNDED REPORTS ONLY.)

COPY NO.	DISTRIBUTION 1. ASSIGN COPY NO. TO HARD COPIES ONLY. 2. LIST MICROFICHE AND ABSTRACT RECIPIENTS LAST. 3. EXTERNAL COPIES: INDICATE TRANSMITTER. 4. CIRCLE COPY NO. OF REPORTS ALREADY DISTRIBUTED.				PUT "X" IN PROPER COLUMNS: EXTERNAL (NON-LAC)	INTERNAL (WITHIN LAC)	TYPE: HARD COPY	TYPE: MICRO FICHE	TYPE: ABSTRACT ONLY	LIBRARY CIRCULATION	NO REVISIONS
MASTER	INDICATE WHERE FILED: ☐ REPORTS SERVICES GROUP ☐ PUBLICATION SERVICES GROUP, _____ PROJECT					x	x				
1	VITAL RECORDS (TO BE TRANSMITTED BY REPORTS SERVICES GROUP)					x	x				
2	REPORTS SERVICES GROUP					x	x				
3, 4	CENTRAL LIBRARY					x	x				
5	A. N. Baker	70-01	63	A-1		X	X				
6	D. L. Bickel	86-11	67	A-1		X	X				
7	B. L. Bivens	80-36	67	A-1		X	X				
8	R. N. Bratkovich	70-10	63	A-1		X	X				
9	P. Brunelli	28-06	146	B-1		X	X				
10	C. A. Burton	70-11	80	A-1		X	X				
11	A. L. Byrnes	75-41	63	A-1		X	X				
12	L. C. Cowgill	75-72	63	A-1		X	X				
13	D. R. Crawford	80-37	67	A-1		X	X				
14	A. R. Curtis	72-71	311	B-6		X	X				
15	R. D. Elliott	75-41	63	A-1		X	X				
16	H. H. Hara	83-01	67	A-1		X	X				

CALAC FORM 5759-5

ENGINEERING REPORT INITIAL DISTRIBUTION LIST

(SEE EPM 4.07)

REPORT NO. LR 28460

TITLE	MODEL	SECURITY CLASS.	DATE
	I.R.	UNCLASSIFIED	7-14-78

APPROVALS

ORIGINATING ORGANIZATION (TITLE & DEPT. NO.)

DIVISION ENGINEER

WO/EWA 21 CLASS 3715 WORK ORDER 5890 E.W.A.

COMMERCIAL ENGINEERING (COMMERCIAL ENGINEERING BRANCH REPORTS)

REMARKS

PRODUCT EVALUATION GROUP

LEGAL BRANCH - PATENT SECTION (STATE ANY RESTRICTIONS)

LIMITATION ON ACCESS TO DATA:

UNLESS LIMITATIONS ON SUBSEQUENT RELEASE OF THIS REPORT ARE STATED BELOW, COPIES WILL BE MADE FREELY ACCESSIBLE TO ALL CORPORATION EMPLOYEES. (IF LIMITED, SUBSEQUENT RELEASE TO OTHER ORGANIZATIONS REQUIRES COMPLETION OF FORM 7229.)

LIMITED TO:

REASON:

DATE ON WHICH LIMITATION MAY BE LIFTED:

WOULD IT BE BENEFICIAL TO CALAC TO RELEASE THIS REPORT TO THE PUBLIC VIA NASA/DoD LIBRARIES? ☐YES ☐NO
(ANSWER THIS QUESTION FOR INDEPENDENT RESEARCH OR INDEPENDENT DEVELOPMENT FUNDED REPORTS ONLY.)

COPY NO.	DISTRIBUTION 1. ASSIGN COPY NO. TO HARD COPIES ONLY. 2. LIST MICROFICHE AND ABSTRACT RECIPIENTS LAST. 3. EXTERNAL COPIES: INDICATE TRANSMITTER. 4. CIRCLE COPY NO. OF REPORTS ALREADY DISTRIBUTED.	PUT "X" IN PROPER COLUMNS: EXTERNAL (NON-LAC)	INTERNAL (WITHIN LAC)	TYPE: HARD COPY	MICRO FICHE	ABSTRACT ONLY	LIBRARY CIRCULATION	NO REVISIONS
MASTER	INDICATE WHERE FILED: ☐REPORTS SERVICES GROUP ☐PUBLICATION SERVICES GROUP, _____ PROJECT		X	X				
17	R. Harris, Jr. (GELAC 87-14 B-1B 274		X	X				
18	D. H. Janda 75-71 90 A-1		X	X				
19	F. W. Johnson 75-73 90 A-1		X	X				
20	T. R. Jones 80-36 67 A-1		X	X				
21	D. K. Kawamoto 80-34 67 A-1		X	X				
22	P. H. Kretsinger 80-36 67 A-1		X	X				
23	J. G. Lewolt 75-71 63 A-1		X	X				
24	R. W. Lingard 80-36 67 A-1		X	X				
25	J. D. Little 80-36 67 A-1		X	X				
26	J. J. Lucas 80-34 67 A-1		X	X				
27	R. F. O'Connell 75-71 63 A-1		X	X				
28	R. B. Ostrom 75-72 63 A-1		X	X				
29	R. R. Plank 70-01 63 A-1		X	X				
30	N. A. Radovcich 75-71 63 A-1		X	X				
31	S. W. Robinson 80-01 67 A-1		X	X				

ENGINEERING REPORT INITIAL DISTRIBUTION LIST

(SEE EPM 4-07)

REPORT NO. LR 28460

PAGE 3 OF 3

TITLE	MODEL	SECURITY CLASS.	DATE
	I.R.	UNCLASSIFIED	7-14-78

APPROVALS

ORIGINATING ORGANIZATION (TITLE & DEPT. NO.)

DIVISION ENGINEER

WO/EWA 21 CLASS 3715 WORK ORDER 5890 E.W.A.

COMMERCIAL ENGINEERING
(COMMERCIAL ENGINEERING BRANCH REPORTS)

REMARKS

PRODUCT EVALUATION GROUP

LEGAL BRANCH - PATENT SECTION (STATE ANY RESTRICTIONS)

LIMITATION ON ACCESS TO DATA:

UNLESS LIMITATIONS ON SUBSEQUENT RELEASE OF THIS REPORT ARE STATED BELOW, COPIES WILL BE MADE FREELY ACCESSIBLE TO ALL CORPORATION EMPLOYEES. (IF LIMITED, SUBSEQUENT RELEASE TO OTHER ORGANIZATIONS REQUIRES COMPLETION OF FORM 7229.)

LIMITED TO:

REASON:

DATE ON WHICH LIMITATION MAY BE LIFTED:

WOULD IT BE BENEFICIAL TO CALAC TO RELEASE THIS REPORT TO THE PUBLIC VIA NASA/DoD LIBRARIES? ☐YES ☐NO
(ANSWER THIS QUESTION FOR INDEPENDENT RESEARCH OR INDEPENDENT DEVELOPMENT FUNDED REPORTS ONLY.)

COPY NO.	DISTRIBUTION 1. ASSIGN COPY NO. TO HARD COPIES ONLY. 2. LIST MICROFICHE AND ABSTRACT RECIPIENTS LAST. 3. EXTERNAL COPIES: INDICATE TRANSMITTER. 4. CIRCLE COPY NO. OF REPORTS ALREADY DISTRIBUTED.				PUT "X" IN PROPER COLUMNS: EXTERNAL (NON-LAC)	INTERNAL (WITHIN LAC)	TYPE: HARD COPY	MICRO FICHE	ABSTRACT ONLY	LIBRARY CIRCULATION	NO REVISIONS
MASTER	INDICATE WHERE FILED: ☐REPORTS SERVICES GROUP ☐PUBLICATION SERVICES GROUP, _____ PROJECT					X	X				
32	D. H. Saiki	80-36	67	A-1		X	X				
33	J. E. Sherman (LMSC)	19-40	102			X	X				
34	G. E. Smith	80-01	67	A-1		X	X				
35	J. F. Stroud	75-42	63	A-1		X	X				
36	F. R. Webster	86-10	67	A-1		X	X				
37	H. P. Weinberger	80-36	67	A-1		X	X				
38	M. L. White	80-36	67	A-1		X	X				
39	NASA VIA REPTS. SERVICES				X						
40	DOD LIBRARY VIA REPTS SERVICES				X						

CALAC FORM 5759-5